# Use-DatabaseReleaseArtifact

## Use-DatabaseReleaseArtifact

Uses the IReleaseArtifact object produced by the New-DatabaseReleaseArtifact cmdlet to update a target database.

### Syntax

```
Use-DatabaseReleaseArtifact [-InputObject] <IReleaseArtifact> -DeployTo <DatabaseConnection> [-
SkipPreUpdateSchemaCheck] [-SkipPostUpdateSchemaCheck] [-QueryBatchTimeout <int>] [-AbortOnWarningLevel
<WarningSeverity>] [-DisableMonitorAnnotation] [-ReleaseUrl <string>] [<CommonParameters>]
```

### Description

The Use-DatabaseReleaseArtifact cmdlet executes the SQL update script inside the database deployment resources produced by the New-DatabaseReleaseArtifact cmdlet. This updates the target database to the same state as the schema specified in the IReleaseArtifact object.

Before executing the script, SQL Change Automation first checks that the target database schema matches the target schema defined in the IReleaseArtifact object. If the schemas don't match, SQL Change Automation will then check if the target database schema matches the source schema specified in the Use-DatabaseReleaseArtifact cmdlet. (This is to check if the update has already been applied and the target schema is already in the desired state). If these schemas also don't match, the cmdlet fails. If the checks show the target schema is already in the desired state, SQL Change Automation will skip running the SQL update script and the cmdlet will finish. To turn off these checks, use the SkipPreUpdateSchemaCheck parameter.

The Use-DatabaseReleaseArtifact cmdlet then executes the SQL script to update the target database. It logs any errors that occur in the execution, but will continue executing the whole script unless the connection to the database is broken (for example, if the database goes offline).

After executing the script, the cmdlet checks that the target database schema has updated correctly to the 'Source' schema specified in the New-DatabaseReleaseArtifact cmdlet. If the schema hasn't updated correctly, the cmdlet returns an error message. To turn off this check use the SkipPostUpdateSchemaCheck parameter.

By default, SQL Change Automation deploys static data contained in a NuGet package, scripts folder or zip file; no data will be deployed from a live data source. The pre-update and post-update schema checks also check for differences in static data. To turn off the deployment of static data and to exclude static data from the schema checks, add IgnoreStaticData to the New-DatabaseReleaseArtifact command. For more information about static data, see http://www.red-gate.com/sca/ps/help/staticdata.

If the database deployment resources contain a filter, this will be applied to the pre-deployment and post-deployment schema checks; only differences between objects included by the filter will cause the check to fail. For more information about filters, see http://www.red-gate.com/sca/ps/help/filters.

The SQL Compare options specified in the IReleaseArtifact object will be used in the schema comparison checks before and after running the update. For more information about SQL Compare options, see http://www.red-gate.com/sca/ps/help/compareoptions.

### Parameters

#### -InputObject <RedGate.Versioning.Automation.Compare.Domain.ReleaseArtifacts.IReleaseArtifact>

The Release Artifact to be deployed.

| | |
|---|---|
| Aliases | None |
| Required? | true |
| Position? | 0 |
| Default Value | None |
| Accept Pipeline Input | true (ByValue) |
| Accept Wildcard Characters | false |

#### -DeployTo <RedGate.Versioning.Automation.Compare.SchemaSources.DatabaseConnection>

A Database Connection object or database connection string that identifies the target database to be updated. See New-DatabaseConnection for details.

| | |
|---|---|
| Aliases | None |
| Required? | true |
| Position? | named |

| | |
|---|---|
| Default Value | None |
| Accept Pipeline Input | false |
| Accept Wildcard Characters | false |

## -SkipPreUpdateSchemaCheck <System.Management.Automation.SwitchParameter>

Before running the update script, don't check that the target database has the correct schema or that the target database has already been updated.

| | |
|---|---|
| Aliases | None |
| Required? | false |
| Position? | named |
| Default Value | False |
| Accept Pipeline Input | false |
| Accept Wildcard Characters | false |

## -SkipPostUpdateSchemaCheck <System.Management.Automation.SwitchParameter>

Don't check that the target database has the correct schema after the update has run.

| | |
|---|---|
| Aliases | None |
| Required? | false |
| Position? | named |
| Default Value | False |
| Accept Pipeline Input | false |
| Accept Wildcard Characters | false |

## -QueryBatchTimeout <System.Int32>

The execution timeout, in seconds, for each batch of queries in the update script. The default value is 30 seconds. A value of zero indicates that no execution timeout will be enforced.

| | |
|---|---|
| Aliases | None |
| Required? | false |
| Position? | named |
| Default Value | 30 |
| Accept Pipeline Input | false |
| Accept Wildcard Characters | false |

## -AbortOnWarningLevel <RedGate.Versioning.Automation.Shared.Domain.WarningSeverity>

Use this parameter to set the minimum warning level that will cause the release execution operation to abort.

Valid warning severity levels are:

- High

- Medium

- Low

- Information

- None (do not abort for any warnings)

The default setting is 'None'.

Possible values: Information, Low, Medium, High, None

| | |
|---|---|
| Aliases | None |

| | |
|---|---|
| Required? | false |
| Position? | named |
| Default Value | None |
| Accept Pipeline Input | false |
| Accept Wildcard Characters | false |

### `-DisableMonitorAnnotation <System.Management.Automation.SwitchParameter>`

Don't add annotations to the event log for SQL Monitor.

| | |
|---|---|
| Aliases | None |
| Required? | false |
| Position? | named |
| Default Value | False |
| Accept Pipeline Input | false |
| Accept Wildcard Characters | false |

### `-ReleaseUrl <System.String>`

A web page to link this release to. Will be included in the SQL Monitor annotation.

| | |
|---|---|
| Aliases | None |
| Required? | false |
| Position? | named |
| Default Value | None |
| Accept Pipeline Input | false |
| Accept Wildcard Characters | false |

### `<CommonParameters>`

This cmdlet supports the common parameters: -Verbose, -Debug, -ErrorAction, -ErrorVariable, -OutBuffer, and -OutVariable. For more information, see http://technet.microsoft.com/en-us/library/hh847884.aspx.

## Inputs

The input type is the type of the objects that you can pipe to the cmdlet.

- **RedGate.Versioning.Automation.Compare.Domain.ReleaseArtifacts.IReleaseArtifact**

  The Release Artifact to be deployed.

## Return values

The output type is the type of the objects that the cmdlet emits.

- **None**

## Examples

### ---------- EXAMPLE 1 ----------

```
$production = New-DatabaseConnection -ServerInstance "prod01\sql2014" -Database "Production" -Username
"AutomationUser" -Password "P@ssw0rd"
$build = Import-DatabaseBuildArtifact "C:\Work\buildArtifacts\DatabaseBuildArtifact.1.0.0.nupkg"

$update = New-DatabaseReleaseArtifact -Source $build -Target $production

Use-DatabaseReleaseArtifact $update -DeployTo $production
```

This example shows how to update a database based on a SQL Change Automation Build Artifact, DatabaseBuildArtifact.1.0.0.nupkg.

This cmdlet runs the targeted deployment script contained in the IReleaseArtifact, which updates the database "Production".

## ---------- EXAMPLE 2 ----------

```
$staging = New-DatabaseConnection -ServerInstance "staging01\sql2014" -Database "Staging" -Username
"AutomationUser" -Password "P@ssw0rd"
$buildArtifact = "C:\Work\buildArtifacts\MyDatabase.1.0.0.nupkg"

$update = New-DatabaseReleaseArtifact -Source $buildArtifact -Target $staging

Use-DatabaseReleaseArtifact $update -DeployTo $staging
```

This example shows how to update a database based on a SQL Source Control Build Artifact, MyDatabase.1.0.0.nupkg.

The New-DatabaseReleaseArtifact cmdlet creates the database deployment resources. It uses the database, Staging, as the Target parameter: this is the schema to be updated. It uses a NuGet package containing a scripts folder, database.nupkg, as the Source parameter: this is the schema that Staging will be updated to.

The IReleaseArtifact object, $update, is then passed to the Use-DatabaseReleaseArtifact cmdlet. This cmdlet runs the SQL update script contained in $update, which updates Staging to the schema contained in database.nupkg.

Before running the SQL update script, Use-DatabaseReleaseArtifact checks that Staging still has the same schema that it did at the time that $update was created. It then runs the script, and afterwards, checks that the update was successful and that Staging has the correct new schema.

If $package contains static data, SQL Change Automation will deploy it to the $staging database. The pre-deployment and post-deployment schema checks also check for differences in static data. To turn off the deployment of static data and to exclude static data from the schema checks, add the IgnoreStaticData parameter to New-DatabaseReleaseArtifact.

## ---------- EXAMPLE 3 ----------

```
$staging = New-DatabaseConnection -ServerInstance "staging01\sql2014" -Database "Staging" -Username
"AutomationUser" -Password "P@ssw0rd"
$buildArtifact = "C:\Work\buildArtifacts\MyDatabase.1.0.0.nupkg"

$update = New-DatabaseReleaseArtifact -Source $buildArtifact -Target $staging

Use-DatabaseReleaseArtifact $update -DeployTo $staging -SkipPostUpdateSchemaCheck
```

This example shows how to skip the post-deployment check that verifies the target database matches the source.

The Use-DatabaseReleaseArtifact cmdlet updates the database, Staging, to match the schema contained in the NuGet package, database.nupkg. The SkipPostUpdateSchemaCheck parameter is used to skip the post-update schema check.

## ---------- EXAMPLE 4 ----------

```
$staging = New-DatabaseConnection -ServerInstance "staging01\sql2014" -Database "Staging" -Username
"AutomationUser" -Password "P@ssw0rd"

Import-DatabaseReleaseArtifact -Path "C:\Work\DatabaseRelease" | Use-DatabaseReleaseArtifact -DeployTo $staging
```

This example shows how to use the Import-DatabaseReleaseArtifact cmdlet to import the database deployment resources from a folder created previously by the Export-DatabaseReleaseArtifact cmdlet. The database, Staging, is updated to the schema specified by the IReleaseArtifact object.